IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2019, No. 2, pp. 193–217. DOI:10.13154/tosc.v2019.i2.193-217

Reconstructing an S-box from its Difference Distribution Table

Orr Dunkelman and Senyang Huang(✉)

Department of Computer Science, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il,shuang@campus.haifa.ac.il

Abstract.
In this paper we study the problem of recovering a secret S-box from its difference distribution table (DDT). While being an interesting theoretical problem on its own, the ability to recover the S-box from the DDT of a secret S-box can be used in cryptanalytic attacks where the attacker can obtain the DDT (e.g., in Bar-On et al.'s attack on GOST), in supporting theoretical analysis of the properties of difference distribution tables (e.g., in Boura et al.'s work), or in some analysis of S-boxes with unknown design criteria (e.g., in Biryukov and Perrin's analysis) .
We show that using the well established relation between the DDT and the linear approximation table (LAT), one can devise an algorithm different from the straightforward guess-and-determine (GD) algorithm proposed by Boura et al. Moreover, we show how to exploit this relation, and embed the knowledge obtained from it in the GD algorithm. We tested our new algorithm on random S-boxes of different sizes, and for random 14-bit bijective S-boxes, our results outperform the GD attack by several orders of magnitude.

Keywords: S-box · DDT · LAT · the sign determination problem

1 Introduction

Differential cryptanalysis, introduced by Biham and Shamir [BS91], has transformed the field of cryptanalysis and offered attacks against multiple symmetric-key primitives (and a few public-key ones). An essential component in estimating the probability of a differential characteristic is the *Difference Distribution Table* of an S-box. This table is easy to compute when the S-box is given (in time $O(2^{2n})$ for an n-bit S-box). However, the inverse problem of deducing the S-box from a given DDT, was mostly left unstudied.

At first, this problem looks like a theoretical problem of very limited practical interest. However, efficient reconstruction of the S-box from the DDT is a useful tool in several cases. First, several cryptanalytic attacks on secret S-boxes constructions (such as GOST [GOS98] and Blowfish [Sch94]) may have access to the difference distribution table rather than the S-box itself. For example, in Bar-On et al.'s slide attack on GOST [BOBDK18], the attacker can learn the DDT, and needs to deduce the secret S-box from it.

Another line of research that will enjoy such efficient reconstruction algorithms is the study of the theoretical properties of DDTs. A recent work by Boura et al. [BCJS19] studied a theoretical question — can two different S-boxes, that do not satisfy some trivial relation, share the same DDT. As part of this work, a guess-and-determine (GD) algorithm for the reconstruction of the S-box was introduced and used.[1] While being practical for

[1]We note that a related algorithm that accepts a DDT and finds an S-box that conforms to the high values of the DDT, was proposed in [BP15]

Received: 2018-09-01, Revised: 2019-03-01, Accepted: 2019-05-01, Published: 2019-06-11

small S-boxes, this algorithm's running time was not analyzed for the general case, and it seems that for large S-boxes it may be impractical.

In this paper we tackle the reconstruction problem using a different approach. We rely on the well-established relation between the DDT of an S-box and the S-box's LAT [BLN17, BN13, CV95]. We show that using this relation, it is possible to transform the DDT into multiple linear approximation tables,[2] each of which is offering an S-box (that can be easily computed relying on the Walsh-Hadamard transform).

More precisely, we first use this relation to reconstruct as many of the Boolean functions $c_i \cdot S(x)$ as we can (and need to) use this relation. For m-bit output S-boxes, reconstructing m such independent Boolean functions, i.e., $c_i \cdot S(x)$ where $0 \leq i < m$ and $0 \leq c_i < 2^m$, is sufficient to trivially and efficiently reconstruct the S-box $S(x)$.

After analyzing the process of the reconstruction of a single $c_i \cdot S(x)$, we show how to use the knowledge obtained to improve the GD algorithm. We offer a heuristic analysis of the running time of both the GD algorithm (which may be of independent interest) and of our approach, suggesting that the combination offers superior results to the previous ones. More precisely, for many types of S-boxes, it is expected that our algorithm outperforms the GD algorithm.

Finally, we test different types of S-boxes, checking the time complexities of the actual reconstruction for different sizes of S-boxes. We compare our method with the simple GD algorithm and discuss in which cases our new method provides better performance than the simple guess and determine attack. For example, for 8-bit to 8-bit S-boxes, it seems that our algorithm is fairly comparable to the standard GD one. However, as the S-box size increases, our approach becomes significantly better – for 10-bit S-boxes, our approach is on average 10 times faster (and also the median is about 10 times better), for 12-bit S-boxes, our approach is about 4,500 times faster on average, and for 14-bit S-boxes, the speed-up is by a factor of $6.8 \cdot 10^6$.

This paper is organized as follows. In Section 2, we discuss the preliminary of the reconstruction problem, including the DDT and LAT, the previous works on the relation between an S-box, its DDT and its LAT. The notations used in this paper are also introduced. In Section 3, the problem of recovering the Boolean function $c_i \cdot S(x)$ is solved by introducing a new problem which we call *the sign determination problem*. With the knowledge obtained by solving the new problem, the GD algorithm of [BCJS19] is improved in Section 4. Then, our approach is tested on different S-boxes and some special Boolean functions. In Section 5 we compare the performances of our method with the GD algorithm of [BCJS19]. In Section 6, we conclude this paper.

2 Background and Notations

Throughout the paper we discuss S-boxes with n-bit inputs and m-bit outputs, i.e., $n \times m$ S-boxes. When $m = n$, we refer to the S-box simply as an n-bit S-box. We treat the S-box as a vectorial Boolean function, i.e., $S(x) = (S^{m-1}(x), \ldots, S^0(x))$, with m Boolean functions $S^i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $0 \leq i < m$.

After recalling the definitions of the difference distribution table and the linear approximation table, the previous work on the relation between them is revisited. We then introduce additional notations which are used in this paper. We then quickly recall some properties of Hadamard matrices.

[2]As we later discuss in Section 2.2, this relation allows for recovering the absolute values of the entries of the LAT, and the signs of the different entries are to be determined.

2.1 Difference Distribution Table and Linear Approximation Table

The *difference distribution table* (DDT) of an S-box counts the number of cases when the input difference of a pair is a and the output difference is b (see [BS91]).[3] For an input difference $a \in \mathbb{F}_2^n$ and an output difference $b \in \mathbb{F}_2^m$, the entry $\delta(a, b)$ of the S-box's DDT is:

$$\delta(a,b) = \left|\{z \in \mathbb{F}_2^n \middle| S(z \oplus a) \oplus S(z) = b\}\right|.$$

In [O'C94, O'C95], O'Connor discussed the DDT of a random bijective n-bit S-box, showing that for $a, b \neq 0$, $\delta(a,b) = 2t$, where $t \sim \text{Poi}(1/2)$. Later, Daemen and Rijmen investigated $n \times m$ S-boxes, reaching related conclusions in Corollary 2 of [DR07], that suggests the probability that a random entry of the DDT is non-zero is

$$P_{n,m}^{DDT} = \Pr[\delta(a,b) \neq 0, a \in \mathbb{F}_2^n \setminus \{0^n\}, b \in \mathbb{F}_2^m] \approx 1 - e^{-2^{n-m-1}}.$$

We follow the work of Boura et al. in [BCJS19] and call two S-boxes $S_0(x)$ and $S_1(x)$ *DDT-equivalent* if they have the same DDT. We also call an element $a \in \mathbb{F}_2^n$ a *linear structure* of the S-box $S(x)$ if $S(x) \oplus S(x \oplus a)$ is constant. In [BCJS19], a DDT-equivalence class is called *trivial* when its size matches the lower-bound given in Property 1:

Property 1. ([BCJS19]) Let S be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ and let ℓ denote the dimension of its linear space, i.e., of the space formed by all linear structures of S. Then, the DDT-equivalence class of S necessarily contains the $2^{m+n-\ell}$ distinct functions of the form

$$x \to S(x \oplus c) \oplus d, \qquad c \in \mathbb{F}_2^n, \quad d \in \mathbb{F}_2^m.$$

The *differential uniformity* is an important characteristic for analysing the resistance to differential cryptanalysis (see [Nyb94]). The differential uniformity of an S-box $S(x)$ is defined as

$$\max_{a \in \mathbb{F}_2^n \setminus \{0^n\}, b \in \mathbb{F}_2^m} \delta(a,b).$$

The lowest possible value for the differential uniformity of a function from $\mathbb{F}_2^n$ into itself is two and functions with differential uniformity two are called almost perfect nonlinear (APN). As we discuss in Section 5, it is harder to reconstruct APN functions with input dimension between 7 and 11 from their DDT compared to random S-boxes, using our technique.

The linear approximation table (LAT) of an S-box is used to derive approximate linear relations between input bits and output bits of the S-box [Mat94]. For any input mask $a \in \mathbb{F}_2^n$ and any output mask $b \in \mathbb{F}_2^m$, the LAT entry is defined as

$$\lambda(a,b) = \left|\{x \in \mathbb{F}_2^n \middle| a \cdot x \oplus b \cdot S(x) = 0\}\right| - 2^{n-1} = \frac{1}{2} \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x \oplus b \cdot S(x)} \qquad (1)$$

where $a \cdot x$ and $b \cdot S(x)$ are the inner product over $\mathbb{F}_2$, e.g. $a \cdot x = \sum_{i=0}^{n-1} a_i \cdot x_i$. In Corollary 6 of [DR07], Daemen and Rijmen also discuss the LAT of a random $n \times m$ S-box, showing that the probability that a random entry of the LAT is non-zero is

$$P_{n,m}^{LAT} = \Pr[\lambda(a,b) \neq 0, a \in \mathbb{F}_2^n \setminus \{0^n\}, b \in \mathbb{F}_2^m] \approx 1 - \frac{1}{\sqrt{2\pi} \cdot 2^{(n-2)/2}}.$$

[3]We note that [BS91] also discusses DDTs that store these pairs. However, for the sake of this work we use the common DDT that stores only the number of pairs.

The *nonlinearity* of a Boolean function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ is the minimal number of truth table entries that must be changed in order to become an affine function. In our case, for each $b \in \mathbb{F}_2^m$, the nonlinearity of $b \cdot S(x)$ is

$$n\ell(b \cdot S(x)) = 2^{n-1} - \max_{a \in \mathbb{F}_2^n \setminus \{0^n\}} |\lambda(a,b)|.$$

Let f be a Boolean function on n variables, where n is even. f is a *bent function* if its nonlinearity is 2^n. When for all a, $|\lambda(a,b)|$ in the b-th column is equal to $2^{n/2-1}$, $b \cdot S(x)$ is a bent function [Car10].

2.2 Links between an S-box, its Difference Distribution Table and its Linear Approximation Table

We now revisit the relation between the DDT and the LAT of an S-box observed in [BLN17, BN13, CV95]. To do so, we start with the Walsh-Hadamard transform. Let $f : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{R}$ be a function. $\hat{f}$ denotes its *Walsh-Hadamard transform*, which is equal to:

$$\hat{f}(a,b) = \sum_{x,y} f(x,y)(-1)^{a \cdot x \oplus b \cdot y}, \tag{2}$$

where $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^m$ and $a \cdot x$ and $b \cdot y$ are the inner product over the domains $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$, respectively. Note that the sum in Equation 2 is evaluated over the reals. Lemma 1 shows that given the S-box's LAT, the attacker can reconstruct the underlying S-box by solving a system of 2^{m+n}-variable linear equations.

Lemma 1. *([CV95, Lemma 2]) For $(a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$, let $\theta(a,b)$ be the characteristic function of S, i.e., $\theta(a,b) = 1$ if and only if $S(a) = b$; otherwise $\theta(a,b) = 0$. Then,*

$$\hat{\lambda}(a,b) = 2^{m+n-1}\theta(a,b).$$

Theorem 1 obtained in [BN13, CV95, DGV95] shows that the entries of the DDT and the LAT are linked to each other through the Walsh-Hadamard transform.

Theorem 1. *([BN13, CV95, DGV95]) For all $(a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$,*

1. $\hat{\delta}(a,b) = 4\lambda^2(a,b)$,
2. $4\widehat{\lambda^2}(a,b) = 2^{m+n}\delta(a,b)$,

where $\widehat{\lambda^2}(a,b)$ is the Walsh transform of $\lambda^2(a,b)$, the squared LAT.

The first conclusion from the above theorem is that given the DDT, one can deduce the squared LAT as follows:

$$\lambda^2(a,b) = \frac{1}{4}\sum_{x,y}(-1)^{a \cdot x \oplus b \cdot y}\delta(x,y). \tag{3}$$

Hence, in order to recover the S-box from the DDT and reconstruct the S-box from the squared LAT, we need to determine the signs of the entries in the squared LAT and reconstruct the S-box from the squared LAT. One can apply a trivial algorithm to reconstruct the S-box by testing all the $2^{P_{n,m}^{LAT} \cdot 2^{(n-1)(m-1)}}$ possibilities for the signs of the non-zero $\lambda(a,b)$ coefficients to recover the LAT. As discussed in Section 3, we introduce a new, and more efficient method, to reconstruct the S-box.

2.3 Notations

We denote the b-th column of the LAT by $\vec{\lambda}_b$, where $0 \leq b < 2^m$. In addition, we use $\lambda^\dagger(a,b)$ to denote the absolute value of the element $\lambda(a,b)$ of the LAT following Equation 3. We extend the $\dagger$ notion to vectors as follows:

$$\vec{v}^{\,\dagger} = (|v_0|, \ldots, |v_{\ell-1}|)^T,$$

where $\vec{v} = (v_0, \ldots, v_{\ell-1})^T$ and $|\cdot|$ is the absolute value of a number. We define the absolute LAT of the vectorial Boolean function S as $(\vec{\lambda}_0^\dagger, \cdots, \vec{\lambda}_{2^m-1}^\dagger)^T$. We also define $\vec{s}_b$ as $((-1)^{b\cdot S(0)}, \ldots, (-1)^{b\cdot S(2^n-1)})^T$. We define for a vector $\vec{v} = (v_0, \ldots, v_{\ell-1})^T$, a partial vector $\vec{v}^{\,[x,y]} = (v_x, \ldots, v_y)^T$, $0 \leq x < y < \ell$. Finally, let $\vec{u} = (u_0, \ldots, u_{\ell-1})^T$ and $\vec{v} = (v_0, \ldots, v_{\ell-1})^T$ be two vectors. Then, the *Hadamard product* of these vectors is denoted by $\vec{u} \odot \vec{v} = (u_0 \cdot v_0, \ldots, u_{\ell-1} \cdot v_{\ell-1})^T$.

2.4 Hadamard Matrices

Let H_n be a $2^n \times 2^n$ Hadamard matrix such that the element in the i-th row, j-th column of H_n is $(-1)^{i\cdot j}$, where $i \cdot j$ is the inner product of i and j for any $0 \leq i, j < 2^n$. We show the recursive definition of these Hadamard matrices as follows:

Definition 1. Let $H_0 = (1)$, then the Hadamard matrix H_i can be represented as

$$H_i = \begin{pmatrix} H_{i-1} & H_{i-1} \\ H_{i-1} & -H_{i-1} \end{pmatrix}, i \geq 1.$$

3 The Sign Determination Problem

As suggested in Section 2.2, given the DDT, we can easily compute $\lambda^\dagger(a,b)$. To recover the S-box we just need to determine the signs of the entries. We define the sign determination problem as follows:

Definition 2. Given $\vec{\lambda}_b^\dagger$ where $1 \leq b < 2^m$, the *sign determination problem* of the b-th column in an LAT is the problem of recovering $\vec{\lambda}_b$ from $\vec{\lambda}_b^\dagger$, i.e., determining the signs of $\lambda(a,b), 0 \leq a < 2^n$.

To solve the sign determination problem, we study the linear relation between $\vec{\lambda}_b$ and $\vec{s}_b$ in Section 3.1. Based on this relation, a basic algorithm for solving the sign determination problem is presented in Section 3.2. In Section 3.3, we observe some interesting properties of the solution space. We use these observations in developing a new and improved algorithm in Section 3.4. We give a tight upper bound of the complexity of the new algorithm in Section 3.5.

3.1 The Linear Relation between $\vec{\lambda}_b$ and $\vec{s}_b$

Property 2. For any b-th column of the linear approximation table (for $0 \leq b < 2^m$), the following formula holds

$$H_n \vec{s}_b = 2\vec{\lambda}_b. \tag{4}$$

Proof. For all $0 \leq p < 2^n$,

$$\begin{aligned} \sum_{a\in\mathbb{F}_2^n} 2(-1)^{a\cdot p}\lambda(a,b) &= \sum_{a\in\mathbb{F}_2^n}\sum_{x\in\mathbb{F}_2^n} (-1)^{a\cdot p} \cdot (-1)^{a\cdot x \oplus b\cdot S(x)} \\ &= \sum_{x\in\mathbb{F}_2^n}\sum_{a\in\mathbb{F}_2^n} (-1)^{b\cdot S(x)}(-1)^{a\cdot x\oplus a\cdot p}. \end{aligned}$$

From Proposition 7 in [Car10], if $x = p$, $\sum_{a\in\mathbb{F}_2^n}(-1)^{a\cdot x\oplus a\cdot p} = 2^n$; otherwise the sum is zero.

$$\sum_{a\in\mathbb{F}_2^n} 2(-1)^{a\cdot p}\lambda(a,b) = (-1)^{b\cdot S(p)}2^n, \quad 0 \le p < 2^n. \tag{5}$$

$$\Rightarrow \qquad H_n\vec{\lambda}_b = 2^{n-1}\vec{s}_b \tag{6}$$

As $H_n \cdot H_n = 2^n I_{2^n}$, this formula can also be written as $H_n\vec{s}_b = 2\vec{\lambda}_b$. Note that when $p = 0$ in Equation 5, it follows that $\sum_{a\in\mathbb{F}_2^n}\lambda(a,b) = \pm 2^{n-1}$. □

The assignments of the b-th column are related to the linear combination of the components of $S(x)$. Let $b = b_{m-1}\dots b_0$ be the binary representation of b. It implies that $b\cdot S(x) = \bigoplus_{i=0}^{m-1} b_i S^i(x)$.

Definition 3. The c_0-th, ..., the c_j-th columns in the LAT where $0 \le c_0 < \dots < c_j < 2^m$ are *independent columns* if the binary representations of $c_0,\dots,c_j$ are linearly independent over $\mathbb{F}_2^m$.

If the attacker solves the sign determination problem for m independent columns, then the attacker can easily recover the S-box. The attacker takes $(c_0,\dots,c_{m-1})$ as an $m\times m$ matrix on $\mathbb{F}_2$, denoted as C. For each $S(i), 1 \le i < 2^n$, let $\vec{s}_i'$ be the vector $(c_0S(i),\dots,c_{m-1}S(i))^T$ over $\mathbb{F}_2^m$, which is known from the solutions of the sign determination problems. The binary representation of $S(i)$, i.e., $(S^0(i),\dots,S^{m-1}(i))$, is obtained immediately by computing $C^{-1}\vec{s}_i'$. Then the S-box can be reconstructed by computing C^{-1} in time $O(m^3)$ and computing $C^{-1}\vec{s}_i'$ for $0 \le i < 2^n$ in time $O(2^n\cdot m^2)$. Thus, the total running time of recovering S from m independent columns is $O(m^3 + m^2\cdot 2^n) \approx O(m^2 2^n)$.

If the attacker applies an exhaustive search to solving the sign determination problem of m independent columns, the complexity of reconstructing the S-box is still very high, which is $O(2^{m\cdot 2^n P_{n,m}^{LAT}} + m^2 2^n)$ as there are m columns of $P_{n,m}^{LAT}2^n$ non-zero elements each. We propose a basic method for solving this problem of one column in Section 3.2 and improve it with a significantly more efficient manner in Section 3.3 and Section 3.4.

3.2 Basic Algorithm for Solving the Sign Determination Problem

3.2.1 Solving the System of Linear Equations $H_n\vec{x} = \vec{y}$

With Definition 1 and the fast Walsh-Hadamard transform [MS77], we can solve the system of linear equations $H_n\vec{x} = \vec{y}$ recursively. By elementary transformation:

$$(H_n,\vec{y}) = \left(\begin{array}{cc|c} H_{n-1} & H_{n-1} & \vec{y}^{[0,2^{n-1}-1]} \\ H_{n-1} & -H_{n-1} & \vec{y}^{[2^{n-1},2^n-1]} \end{array}\right)$$

$$\Rightarrow \left(\begin{array}{cc|c} H_{n-1} & 0 & (\vec{y}^{[0,2^{n-1}-1]} + \vec{y}^{[2^{n-1},2^n-1]})/2 \\ 0 & H_{n-1} & (\vec{y}^{[0,2^{n-1}-1]} - \vec{y}^{[2^{n-1},2^n-1]})/2 \end{array}\right).$$

It is easy to see that the original problem is divided into two independent subproblems as follows:

$$H_{n-1}\vec{x}^{[0,2^{n-1}-1]} = (\vec{y}^{[0,2^{n-1}-1]} + \vec{y}^{[2^{n-1},2^n-1]})/2$$
$$H_{n-1}\vec{x}^{[2^{n-1},2^n-1]} = (\vec{y}^{[0,2^{n-1}-1]} - \vec{y}^{[2^{n-1},2^n-1]})/2.$$

We can recursively apply the above process to the problems in the ℓ-th step. At the beginning of the ℓ-th step, there are $2^{\ell-1}$ problems with $2^{n-\ell+1}$ constraints, denoted as:

$$\begin{aligned} H_{n-\ell+1}\vec{x}^{[0,2^{n-\ell+1}-1]} &= \vec{\beta}_0, \\ &\vdots \\ H_{n-\ell+1}\vec{x}^{[2^n-2^{n-\ell+1},2^n-1]} &= \vec{\beta}_{2^{\ell-1}-1}, \end{aligned} \tag{7}$$

where $\vec{\beta}_0, \ldots, \vec{\beta}_{2^{\ell-1}-1}$ are the vectors obtained from the last step and $1 \le \ell \le n$. Each problem in Equation 7 is divided into two subproblems as follows:

$$\begin{aligned} H_{n-\ell}\vec{x}^{[0,2^{n-\ell}-1]} = \vec{\gamma}_0, &\quad H_{n-\ell}\vec{x}^{[2^{n-\ell},2^{n-\ell+1}-1]} = \vec{\gamma}_1, \\ &\vdots \\ H_{n-\ell}\vec{x}^{[2^n-2^{n-\ell+1},2^n-2^{n-\ell}-1]} = \vec{\gamma}_{2^\ell-2}, &\quad H_{n-\ell}\vec{x}^{[2^n-2^{n-\ell},2^n-1]} = \vec{\gamma}_{2^\ell-1}. \end{aligned} \tag{8}$$

where,

$$\begin{aligned} \vec{\gamma}_0 = \left(\vec{\beta}_0^{[0,2^{n-\ell}-1]} + \vec{\beta}_0^{[2^{n-\ell},2^{n-\ell+1}-1]}\right)/2 &\quad , \quad \vec{\gamma}_1 = \left(\vec{\beta}_0^{[0,2^{n-\ell}-1]} - \vec{\beta}_0^{[2^{n-\ell},2^{n-\ell+1}-1]}\right)/2 \\ &\vdots \\ \vec{\gamma}_{2^\ell-2} = \left(\vec{\beta}_{2^{\ell-1}-1}^{[0,2^{n-\ell}-1]} + \vec{\beta}_{2^{\ell-1}-1}^{[2^{n-\ell},2^{n-\ell+1}-1]}\right)/2 &\quad , \quad \vec{\gamma}_{2^\ell-1} = \left(\vec{\beta}_{2^{\ell-1}-1}^{[0,2^{n-\ell}-1]} - \vec{\beta}_{2^{\ell-1}-1}^{[2^{n-\ell},2^{n-\ell+1}-1]}\right)/2 \end{aligned}$$

The total number of subproblems after the ℓ-th step is 2^ℓ and the number of constraints in each subproblem is $2^{n-\ell}$. At the n-th step, the coefficient matrix in the subproblems is $H_0 = 1$. Thus, the entries of $\vec{x}$ are directly obtained.

3.2.2 The Main Idea

We propose to solve the sign determination problem using a recursive procedure. In each layer, the algorithm combines the linear equations in the problem of the current layer, apply the idea of solving the system of linear equations $H_n\vec{x} = \vec{y}$ to reduce the problem into two independent subproblems and check the consistency of the subproblems. That is the algorithm works on the systems of linear equations with the size reduced by half compared to the ones in the previous layer. Finally, when it reaches the n-th layer, the algorithm returns the solutions to the sign determination problem.

The algorithm can be represented by a tree structure. For ease of explanation we denote the ℓ-th layer of the recursive tree by T_ℓ. The algorithm is initialized by guessing the signs of $\lambda(i,b)$ if $\lambda(i,b) \ne 0$. Thus, the leaf node $T_0[i]$ is assigned with $\{2\lambda^\dagger(i,b), -2\lambda^\dagger(i,b)\}$, for $0 \le i < 2^n$. At the beginning of the ℓ-th layer, the subproblems in Equation 7 are recorded in $T_{\ell-1}$. The i-th constraint in Equation 7 is stored in a vector as $\vec{v}^T = (\vec{\beta}_0[i], \ldots, \vec{\beta}_{2^{\ell-1}-1}[i])$, $0 \le i < 2^{n-\ell+1}$. We call the set a *full set* which contains all the possible i-th constraints in Equation 7, denoted by $F_{\ell-1}[i]$, for $0 \le i < 2^{n-\ell+1}, 1 \le \ell \le (n+1)$. In the basic algorithm, we record the full set $F_{\ell-1}[i]$ in the internal node $T_{\ell-1}[i]$. (This strategy will be replaced by a more effective manner described in Section 3.3 and Section 3.4.)

In the ℓ-th layer, the i-th possible constraints of the new subproblems in Equation 8 are deduced from Equation 7 to construct $F_\ell[i]$, $0 \le i < 2^{n-\ell}$. To do so, for each vector in $\vec{p} \in F_{\ell-1}[i]$ and $\vec{q} \in F_{\ell-1}[i + 2^{n-\ell}]$, a new vector which is defined as $E_{\ell-1}(\vec{p}, \vec{q})$ is computed as described below:

$$\begin{aligned} E_{\ell-1}(\vec{p}, \vec{q})^T =& ((p_0 + q_0)/2, (p_0 - q_0)/2, \ldots, \\ & (p_{2^{\ell-1}-1} + q_{2^{\ell-1}-1})/2, (p_{2^{\ell-1}-1} - q_{2^{\ell-1}-1})/2), \end{aligned}$$

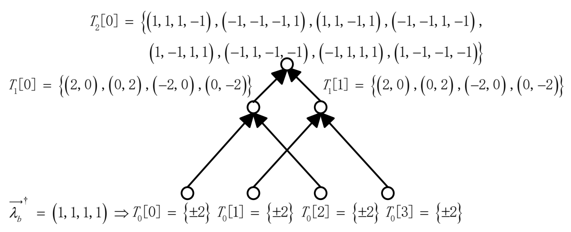
$T_2[0] = \{(1,1,1,-1),(-1,-1,-1,1),(1,1,-1,1),(-1,-1,1,-1),$
$(1,-1,1,1),(-1,1,-1,-1),(-1,1,1,1),(1,-1,-1,-1)\}$
$T_1[0] = \{(2,0),(0,2),(-2,0),(0,-2)\}$
$T_1[1] = \{(2,0),(0,2),(-2,0),(0,-2)\}$
$\vec{\lambda}_b^{\dagger} = (1,1,1,1) \Rightarrow T_0[0] = \{\pm 2\}$ $T_0[1] = \{\pm 2\}$ $T_0[2] = \{\pm 2\}$ $T_0[3] = \{\pm 2\}$

Figure 1: The Tree Structure for $n = 2$

where p_j and q_j are the j-th entries with respect to $\vec{p}$ and $\vec{q}$ and $0 \leq j < 2^{\ell-1}$.

It can be seen from Equation 8 that each entry of the vector in $F_\ell[i]$ is an even number in the range of $-2^{n-\ell}$ and $2^{n-\ell}$ when $1 \leq \ell < n$. As the components of $\vec{s}_b$ are 1 or -1, then in the n-th layer, the entries of the vectors in $F_n[0]$ take their values from the set $\{1, -1\}$. If the constraints over the elements of the vectors in $F_\ell[i]$ are satisfied for the vector $E_{\ell-1}(\vec{p}, \vec{q})$, the new vector is a possible i-th constraint of the new subproblems in Equation 8; otherwise, it should be discarded. When it reaches the n-th layer, the solutions of the sign determination problem are the vectors in the root node $T_n[0]$.

To illustrate our idea more intuitively, we refer to the recursive tree for $n = 2$ in Figure 1 and show an example when $\vec{\lambda}_b^{\dagger} = (1, 1, 1, 1)$ and the corresponding LAT column $\vec{\lambda}_b$ is $(1, 1, 1, -1)$ and $\vec{s}_b = (1, 1, 1, -1)$.[4] The nodes in T_0 are initialized by $T_0[0] = T_0[1] = T_0[2] = T_0[3] = \{\pm 2\}$. As shown in Figure 1, $T_1[0]$ is constructed from $T_0[0]$ and $T_0[2]$ and $T_1[1]$ is from $T_0[1]$ and $T_0[3]$. $T_1[0] = T_1[1] = \{(2, 0), (0, 2), (-2, 0), (0, -2)\}$. Similarly, $T_2[0]$ is built from $T_1[0]$ and $T_1[1]$. For each $\vec{p} \in T_1[0]$ and $\vec{q} \in T_1[1]$, we compute $E_1(\vec{p}, \vec{q})$. For example, when $\vec{p} = \vec{q} = (2, 0)$, $E_1(\vec{p}, \vec{q}) = (2, 0, 0, 0) \notin T_2[0]$. In the end, there are eight vectors in $T_2[0]$, which are $(1, 1, 1, -1)$, $(-1, -1, -1, 1)$, $(1, 1, -1, 1)$, $(-1, -1, 1, -1)$, $(1, -1, 1, 1)$, $(-1, 1, -1, -1)$, $(-1, 1, 1, 1)$ and $(1, -1, -1, -1)$. It can be seen that $\vec{s}_b \in T_2[0]$. We give the pseudo code of the basic algorithm in Algorithm 1.

Similarly to the GD algorithm, we fix $S(0)$ to 0 (or any other constant) to find one representative of the DDT-equivalence class and other DDT-equivalent S-boxes can be obtained applying simple linear transformations based on Property 1. Therefore, Algorithm 1 only returns the vectors with the first element as $(-1)^{b \cdot S(0)} = 1$.

3.3 Observing the Structure in the Full Set and Introducing the Compact Set

When we examine the full set $F_\ell[i]$, we notice that its vectors are related. We can use this relation to offer a more compact representation of $F_\ell[i]$ without losing any solutions. This new compact representation reduces both the time and memory complexities of the search algorithm. In the following, we discuss the structure of the full set first. Then, we improve the basic algorithm of Section 3.2 using the compact representation of the full set.

3.3.1 The Structure of the Full Set

Before presenting the structure of $F_\ell[i]$, we define a set of symmetric permutations.

Definition 4. Let $\vec{v}^T$ be $(v_0, \ldots, v_{2^\ell - 1})$. For $0 \leq j < \ell$, we define π_j^ℓ as follows:

$$\pi_j^\ell(\vec{v}) = (v_{2^j}, \ldots, v_{2^{j+1}-1}, v_0, \ldots, v_{2^j-1}, \ldots,$$
$$v_{2^\ell - 2^j}, \ldots, v_{2^\ell - 1}, v_{2^\ell - 2^{j+1}}, \ldots, v_{2^\ell - 2^j - 1})^T.$$

[4]The entry of the LAT can be odd because of the coefficient $\frac{1}{2}$ of Equation 1.

Algorithm 1 Basic Algorithm for Solving the Sign Determination Problem

```
1: **Input:** $\vec{\lambda}_b^\dagger$;
2: **Output:** $F = \{\vec{u} | H_n\vec{u} = 2\vec{\lambda}_b, \vec{u}[0] = 1\}$
3: for each integer $i \in [0, 2^n - 1]$ do
4:     $F_0[i] = \{2\lambda^\dagger(i,b), -2\lambda^\dagger(i,b)\}$        ▷Initial phase
5: end for
6: $F_n[0] = \text{LAYER}(F_0, 0)$
7: return $F = \{\vec{u} | \vec{u} \in F_n[0], \vec{u}[0] = 1\}$.
8:
9: procedure LAYER($F_\ell$, $\ell$);
10:     for each integer $i \in [0, 2^{n-\ell-1} - 1]$ do
11:         if there are no vectors in $F_\ell[i]$ or $F_\ell[i + 2^{n-\ell-1}]$ then
12:             return There exist no S-boxes corresponding to the given DDT!
13:         end if
14:         $F_{\ell+1}[i] = \emptyset$
15:         for each $\vec{u}$ in $F_\ell[i]$ and each $\vec{v}$ in $F_\ell[i + 2^{n-\ell-1}]$ do
16:             Compute $\vec{w} = E_\ell(\vec{u}, \vec{v})$
17:             if $\ell < n$ then
18:                 if every entry in $\vec{w}$ is even and ranges from $-2^{n-\ell-1}$ to $2^{n-\ell-1}$ then
19:                     $F_{\ell+1}[i] = F_{\ell+1}[i] \cup \{\vec{w}\}$
20:                 end if
21:             else
22:                 if every entry in $\vec{w}$ is 1 or $-1$ then                    ▷ when $\ell = n$
23:                     $F_n[i] = F_n[i] \cup \{\vec{w}\}$
24:                 end if
25:             end if
26:         end for
27:     end for
28:     if $\ell < n$ then
29:         LAYER($F_{\ell+1}$,$\ell + 1$)
30:     else
31:         return $F_n[0]$
32:     end if
33: end procedure
```

Note that the permutation π_j^ℓ swaps every two consecutive blocks of 2^j elements in $\vec{v}$ pairwise. Let Π_ℓ be a set of symmetric permutations $\pi_0^\ell, \ldots, \pi_{\ell-1}^\ell$. It can be easily verified that each permutation in Π_ℓ is of order two and that $\pi_0^\ell, \ldots, \pi_{\ell-1}^\ell$ are pairwise commutative. Suppose that $\vec{v} = E_{\ell-1}(\vec{p}, \vec{q})$, the vectors which generate $\pi_j^\ell(\vec{v})$ for $0 \le j < \ell$ in the $(\ell-1)$-th layer are shown in Lemma 2, .

Lemma 2. *When* $0 < j < \ell$, $\pi_j^\ell(\vec{v}) = E_{\ell-1}(\pi_{j-1}^{\ell-1}(\vec{p}), \pi_{j-1}^{\ell-1}(\vec{q}))$*; when* $j = 0$, $\pi_0^\ell(\vec{v}) = E_{\ell-1}(\vec{p}, -\vec{q})$.

Proof. By the definition of operation $E_{\ell-1}$,

$$\begin{aligned} \vec{p}^{\,T} &= (v_0 + v_1, v_2 + v_3, \ldots, v_{2^{\ell+1}-2} + v_{2^{\ell+1}-1}) \\ \vec{q}^{\,T} &= (v_0 - v_1, v_2 - v_3, \ldots, v_{2^{\ell+1}-2} - v_{2^{\ell+1}-1}). \end{aligned}$$

When $1 \leq j < \ell$, the vectors that generate $\pi_j^\ell(\vec{v})$ are

$$\begin{aligned}\vec{p'} = (&v_{2^j} + v_{2^j+1}, \ldots, v_{2^{j+1}-2} + v_{2^{j+1}-1}, v_0 + v_1, \ldots, v_{2^j-2} + v_{2^j-1}, \ldots,\\ &v_{2^\ell-2^j} + v_{2^\ell-2^j+1}, \ldots, v_{2^\ell-2} + v_{2^\ell-1},\\ &v_{2^\ell-2^{j+1}} + v_{2^\ell-2^{j+1}+1}, \ldots, v_{2^\ell-2^j-2} + v_{2^\ell-2^j-1})^T,\\ \vec{q'} = (&v_{2^j} - v_{2^j+1}, \ldots, v_{2^{j+1}-2} - v_{2^{j+1}-1}, v_0 - v_1, \ldots, v_{2^j-2} - v_{2^j-1}, \ldots,\\ &v_{2^\ell-2^j} - v_{2^\ell-2^j+1}, \ldots, v_{2^\ell-2} - v_{2^\ell-1},\\ &v_{2^\ell-2^{j+1}} - v_{2^\ell-2^{j+1}+1}, \ldots, v_{2^\ell-2^j-2} - v_{2^\ell-2^j-1})^T.\end{aligned}$$

It follows from the definition of π_j^ℓ that $\vec{p'} = \pi_{j-1}^{\ell-1}(\vec{p})$ and $\vec{q'} = \pi_{j-1}^{\ell-1}(\vec{q})$. Thus, $\pi_j^\ell(\vec{v}) = E_{\ell-1}(\pi_{j-1}^{\ell-1}(\vec{p}), \pi_{j-1}^{\ell-1}(\vec{q}))$. For the case when $j = 0$, it can be easily verified that $\pi_0^\ell(\vec{v}) = E_{\ell-1}(\vec{p}, -\vec{q})$. □

Now, we define a j-symmetric relation between two vectors $\vec{u}$ and $\vec{v}$ with respect to the permutations in Π_ℓ that helps in capturing the structure of the full set $F_\ell[i]$.

Definition 5. For each $0 \leq j < \ell$, the vector $\vec{u}$ is *j-symmetric* to the vector $\vec{v}$ if there exist $p \geq 1$ permutations $\pi_{j_0}^\ell, \ldots, \pi_{j_{p-1}}^\ell \in \Pi_\ell$ such that

$$\vec{u} = \pm \pi_{j_{p-1}}^\ell \circ \ldots \circ \pi_{j_0}^\ell(\vec{v}),$$

where $0 \leq j_0 < \cdots < j_{p-1} = j$. For the special case when $j = \ell$, the *ℓ-symmetric vectors* to $\vec{u}$ are defined as $\vec{u}$ and $-\vec{u}$. If $\vec{u} = \pi_{j_{p-1}}^\ell \circ \ldots \circ \pi_{j_0}^\ell(\vec{v})$, $\vec{u}$ is *positive j-symmetric* to $\vec{v}$.

We say that $\vec{u}$ is symmetric-equivalent to $\vec{v}$ if for some j, $\vec{u}$ is j-symmetric to $\vec{v}$. It can be easily verified that the symmetric-equivalent relation is an equivalence relation. Moreover, the set of vectors that are positive j-symmetric to $\vec{u}$ for some j is denoted by $[\vec{u}]^+$. For any $0 \leq j \leq \ell$, we denote the set of vectors that are j-symmetric to $\vec{u}$ as $[\vec{u}]_j$. Thus, for each vector $\vec{u} \in F_\ell[i]$, the equivalence class of $\vec{u}$ is $[\vec{u}] = \bigcup_{j=0}^{\ell} [\vec{u}]_j$. In Theorem 2, we present the symmetric structure of $F_\ell[i]$.

Theorem 2. *For any vector $\vec{u} \in F_\ell[i]$ and for any $0 \leq i < 2^{n-\ell}$ and $0 \leq j \leq \ell$, if a vector $\vec{v}$ is j-symmetric to $\vec{u}$, then $\vec{v} \in F_\ell[i]$.*

Proof. Let us consider the case when a vector $\vec{v}$ is j-symmetric to $\vec{u}$, $0 \leq j < l$. For the positive case in Definition 5, we first prove inductively that for each $j' \leq j$, $\pi_{j'}^\ell(\vec{v}) \in F_\ell[i]$. The negative case in Definition 5 can be proved with a similar method.

The statement is true when $j' = 0$. It follows from Lemma 2 that if $\vec{v} = E_{\ell-1}(\vec{p}, \vec{q})$, then $\pi_0^\ell(\vec{v}) = E_{\ell-1}(\vec{p}, -\vec{q})$. As $-\vec{q} \in F_{\ell-1}[i + 2^{n-\ell+1}]$, $\pi_0^\ell(\vec{v}) \in F_\ell[i]$. Assume that when $j' = k$, $\pi_k^\ell(\vec{v}) \in F_\ell[i]$. When $j' = k+1$, it can be seen that $\pi_{k+1}^\ell(\vec{v}) \in F_\ell[i]$. From Lemma 2, if $\vec{v} = E_{\ell-1}(\vec{p}, \vec{q})$, $\pi_{k+1}^\ell(\vec{v}) = E_{\ell-1}(\pi_k^{\ell-1}(\vec{p}), \pi_k^{\ell-1}(\vec{q}))$. As $\vec{p} \in F_{\ell-1}[i]$ and $\vec{q} \in F_{\ell-1}[i + 2^{n-\ell+1}]$, it follows from the assumption directly that $\pi_k^{\ell-1}(\vec{p}) \in F_{\ell-1}[i]$ and $\pi_k^{\ell-1}(\vec{q}) \in F_{\ell-1}[i+2^{n-\ell+1}]$. It can be concluded that $\pi_{k+1}^\ell(\vec{v})$ is in $F_\ell[i]$. Therefore, it can be observed that if $\vec{v} \in F_\ell[i]$, $\pi_{j_{p-1}}^\ell \circ \ldots \circ \pi_{j_0}^\ell(\vec{v}) \in F_\ell[i]$, for any $0 \leq j_0 < \cdots < j_{p-1} = j$, $p \geq 1$.

For the case when $j = \ell$, the positive case is trivial as $\vec{u} = \vec{v}$. The negative case is proved inductively. When $\ell = 0$, the statement is true: if $\lambda^\dagger(i, b) \neq 0$, $F_0[i] = \{\lambda^\dagger(i, b), -\lambda^\dagger(i, b)\}$; otherwise, $F_0[i] = \{0\}$. Assume that the proposition holds when $\ell = k$. When ℓ takes $k+1$, if $\vec{v} = E_k(\vec{p}, \vec{q})$, then $-\vec{v} = E_k(-\vec{p}, -\vec{q})$, where $\vec{p} \in F_k[i]$ and $\vec{q} \in F_k[i + 2^{n-k}]$. From the assumption, $-\vec{p}$ is in $F_k[i]$ and $-\vec{q}$ is in $F_k[i + 2^{n-k}]$. Thus, $-\vec{v}$ is in $F_{k+1}[i]$. □

Now we define the self-j-symmetric vector and the self-j-symmetric set, respectively.

Definition 6. A vector $\vec{u} \in F_\ell[i]$ is *self-j-symmetric* if $\vec{u}$ is j-symmetric to itself, $0 \leq i < 2^{n-\ell}, 0 \leq j < \ell$; a set $F_\ell[i]$ is *self-j-symmetric* if all the vectors in $F_\ell[i]$ are self-j-symmetric.

It is not necessary to check all the vectors in the full set $F_\ell[i]$ to detect whether $F_\ell[i]$ is j-symmetric. We show in Theorem 3 that one can detect whether $F_\ell[i]$ is self-j-symmetric by testing an arbitrary vector in $F_\ell[i]$.

Theorem 3. *For all $1 \leq \ell \leq n$, $0 \leq j < \ell$ and $0 \leq i < 2^{n-\ell}$, $\vec{u} \in F_\ell[i]$ is self-j-symmetric if and only if $F_\ell[i]$ is self-j-symmetric.*

Proof. We define the statement of Theorem 3 as $D(j, \ell)$. We only prove the positive case in Definition 5 inductively for j and ℓ. The negative case can also be proved using a similar method. Let $\vec{p} \in F_{\ell-1}[i]$ and $\vec{q} \in F_{\ell-1}[i + 2^{n-\ell}]$ be the vectors such that $\vec{u} = E_\ell(\vec{p}, \vec{q})$.

We claim that $D(0, \ell)$ is true for $j = 0$ and $1 \leq \ell \leq n$. If $\vec{u} \in F_\ell[i]$ is a self-0-symmetric vector, it indicates that $\vec{u} = (u_0, u_0, \ldots, u_{2^{\ell-1}-1}, u_{2^{\ell-1}-1})$. The vector in $F_{\ell-1}[i + 2^{n-\ell+1}]$ that generate $\vec{u}$ is the zero vector. Let us trace back to the initial values in the 0-th layer which generate the zero vector in $F_{\ell-1}[i + 2^{n-\ell+1}]$. These initial values are zero and the zero vector is the only vector in $F_{\ell-1}[i + 2^{n-\ell+1}]$. It can be seen that for each vector $\vec{v} \in F_\ell[i]$, $\vec{v}$ can be presented as $E_{\ell-1}(\vec{p}, \vec{0})$, where $\vec{p}$ is in $F_{\ell-1}[i]$. It can be verified that $\vec{v}$ is a self-0-symmetric vector.

Suppose that the statement $D(r, k)$ holds for each $r < k$. Then, it can be proved that for $D(r + 1, k + 1)$ is also true. In this case $\vec{u} \in F_{k+1}[i]$ is an self-$(r + 1)$-symmetric vector. There are two scenarios in this situation.

The first scenario is when there exist p permutations in Π_{k+1} such that $\pi^{k+1}_{j_{p-1}} \circ \ldots \circ \pi^{k+1}_{j_1} \circ \pi^{k+1}_{j_0}(\vec{u}) = \vec{u}$, where $p \geq 1$ and $0 = j_0 < j_1 < \cdots < j_{p-1} = r + 1$. Then, $\pi^k_r \circ \ldots \circ \pi^k_{j_1-1}(\vec{p}) = \vec{p}$ and $\pi^k_r \circ \ldots \circ \pi^k_{j_1-1}(\vec{q}) = -\vec{q}$, which indicates that $\vec{p}$ and $\vec{q}$ are self-r-symmetric vectors. It can be concluded that each vector in $F_k[i]$ and $F_k[i + 2^{n-k+1}]$ are self-r-symmetric. Thus, each vector in $F_{k+1}[i]$ is self-$(r + 1)$-symmetric vector.

The second scenario is when there exist p permutations in Π_{k+1} such that $\pi^{k+1}_{j_{p-1}} \circ \ldots \circ \pi^{k+1}_{j_1} \circ \pi^{k+1}_{j_0}(\vec{u}) = \vec{u}$, where $p \geq 1$ and $0 < j_0 < j_1 < \cdots < j_{p-1} = r + 1$. It can be seen that $\pi^k_r \circ \ldots \circ \pi^k_{j_0-1}(\vec{p}) = \vec{p}$ and $\pi^k_r \circ \ldots \circ \pi^k_{j_0-1}(\vec{q}) = \vec{q}$. It can also be concluded that the vectors in $F_{k+1}[i]$ are self-$(r + 1)$-symmetric vectors.

As mentioned above, the statement is true for $j = 0$ and each $1 \leq \ell \leq n$. For each j and ℓ such that $0 \leq j < \ell \leq n$, starting from the statement $D(0, \ell - j)$, $D(j, \ell)$ can be proved by applying the inductive process above. □

3.3.2 Compact Set

Based on Theorem 2, we define a *compact set* $C_\ell[i]$ to be a compact representation of the full set $F_\ell[i]$. $C_\ell[i]$ is a set of representatives of the equivalence classes in $F_\ell[i]$. The relation between the full set $F_\ell[i]$ and its compact set $C_\ell[i]$ is thus:

$$F_\ell[i] = \bigcup_{\vec{u} \in C_\ell[i]} [\vec{u}]. \tag{9}$$

For each vector $\vec{u} \in C_\ell[i]$, any vector $\vec{v} \in [\vec{u}]$ can be constructed using Definition 5. The full set $F_\ell[i]$ can thus be obtained by applying Equation 9 to all $\vec{u} \in C_\ell[i]$, i.e., by computing $[\vec{u}]$ from $\vec{u}$. The compact set $C_\ell[i]$ thus allows rebuilding $F_\ell[i]$ efficiently.

The basic algorithm in Section 3.2 can be optimized by storing the compact set $C_\ell[i]$ instead of the full set $F_\ell[i]$ in the intermediate node $T_\ell[i]$. By doing this, the repeated computation is avoided and the memory consumption is greatly reduced compared to the basic algorithm, as we will show in Section 3.4.

We now propose a technique to construct the compact set $C_{\ell+1}[i]$ from $C_\ell[i]$ and $C_\ell[i + 2^{n-\ell-1}]$, $0 \leq i < 2^{n-\ell-1}$. To build the compact set $C_{\ell+1}[i]$, we construct *a middle* *set* $M_{\vec{u},\vec{w}}$ for each $\vec{u} \in C_\ell[i]$ and $\vec{w} \in C_\ell[i+2^{n-\ell-1}]$ such that $\{\vec{w}\} \subseteq M_{\vec{u},\vec{w}} \subseteq [\vec{w}]$. In this way, the compact set $C_{\ell+1}[i]$ is indeed constructed by computing $E_\ell(\vec{u},\vec{v})$ for each $\vec{u} \in C_\ell[i]$ and $\vec{v}$ in each $M_{\vec{u},\vec{w}}$ such that every two elements in $C_{\ell+1}[i]$ are not j-symmetric to each other, $0 \le j \le \ell$.

The process of building the middle set $M_{\vec{u},\vec{w}}$ is shown in Algorithm 2. The structure of the middle set $M_{\vec{u},\vec{w}}$ is related to the symmetric property of the compact set $C_\ell[i]$. Let $J = \{j \big| C_\ell[i] \text{ is self-}j\text{-symmetric}, 0 \le j < \ell\}$. Note that when J is empty, $M_{\vec{u},\vec{w}} = [\vec{w}]^+$.

Algorithm 2 Constructing $M_{\vec{u},\vec{w}}$ from $\vec{u} \in C_\ell[i]$ and $\vec{w} \in C_\ell[i+2^{n-\ell-1}]$

```
1: procedure ConstructSet($\vec{u}$,$[\vec{w}]^+$, $J$)
2:     $M_{\vec{u},\vec{w}} = [\vec{w}]^+$
3:     for all integers $j \in J$ do
4:         Find $\pi^\ell_{j_0}, \ldots, \pi^\ell_{j_{p-1}}$ such that $\vec{u} = \pm\pi^\ell_{j_{p-1}} \circ \ldots \circ \pi^\ell_{j_0}(\vec{u})$
5:         for all the distinct vectors $\vec{e}, \vec{f}$ in $M_{\vec{u},\vec{w}}$ do
6:             if $\vec{e} = \pm\pi^\ell_{j_{p-1}} \circ \ldots \circ \pi^\ell_{j_0}(\vec{f})$ then
7:                 $M_{\vec{u},\vec{w}} = M_{\vec{u},\vec{w}} \backslash \{\vec{f}\}$
8:             end if
9:         end for
10:    end for
11:    return $M_{\vec{u},\vec{w}}$
12: end procedure
```

It can be seen from Algorithm 2 that the middle set $M_{\vec{u},\vec{w}}$ is constructed by discarding the irrelevant elements from the set $[\vec{w}]^+$, i.e., the vectors in $[\vec{w}]^+$ that generate j-symmetric vectors for some j are carefully selected. Next, we discuss in which form the vectors need to be removed from the set $[\vec{w}]^+$ to build the middle set $M_{\vec{u},\vec{w}}$. Now we show in Lemma 3 that $M_{\vec{u},\vec{w}} \subseteq [\vec{w}]^+$.

Lemma 3. *For each non-zero vector $\vec{v} \in M_{\vec{u},\vec{w}}$, it holds that $-\vec{v} \notin M_{\vec{u},\vec{w}}$.*

Proof. Assume that $-\vec{v} \in M_{\vec{u},\vec{w}}$. Let $\vec{q} = E_\ell(\vec{u},\vec{v})$. From Lemma 2, $E_\ell(\vec{u},-\vec{v}) = \pi_0^{\ell+1}(q) \in C_{\ell+1}[i]$, which reaches a contradiction that each two vectors in $C_{\ell+1}[i]$ are not 0-symmetric. Thus, $-\vec{v} \notin M_{\vec{u},\vec{w}}$. Let $[\vec{w}]^+$ be the set which contains the vectors that are positive j-symmetric to $\vec{w}$ for all $0 \le j \le \ell$. It can be concluded that $M_{\vec{u},\vec{w}} \subseteq [\vec{w}]^+$. ☐

The structure of $M_{\vec{u},\vec{w}}$ is also related to the symmetric property of the vector $\vec{u}$. Suppose that $\vec{u}$ is self-j-symmetric, then there exist p permutations $\pi^\ell_{j_0}, \ldots, \pi^\ell_{j_{p-1}}$ such that $\vec{u} = \pi^\ell_{j_{p-1}} \circ \ldots \circ \pi^\ell_{j_0}(\vec{u})$.[5] For a vector $\vec{v} \in M_{\vec{u},\vec{w}}$, let $\vec{e}$ denote the vector $E_\ell(\vec{u},\vec{v})$, which is in $C_{\ell+1}[i]$. Suppose that $\vec{d} = \pm\pi^\ell_{j_{p-1}} \circ \ldots \circ \pi^\ell_{j_0}(\vec{v}) \in M_{\vec{u},\vec{w}}$ and $\vec{d} \neq \vec{v}$, then $\vec{f} = E_\ell(\vec{u},\vec{d}) \in C_{\ell+1}[i]$. It can be seen from Lemma 2 that $\vec{f} = \pi^{\ell+1}_{j_{p-1}+1} \circ \ldots \circ \pi^{\ell+1}_{j_0+1}(E_\ell(\vec{u},\pm\vec{v}))$, which is self-$(j+1)$-symmetric to $\vec{e}$. This contradicts the fact that each two vectors in $C_{\ell+1}[i]$ are not $(j+1)$-symmetric. Thus, for each $\vec{v} \in M_{\vec{u},\vec{w}}$, $\pm\pi^\ell_{j_{p-1}} \circ \ldots \circ \pi^\ell_{j_0}(\vec{v}) \notin M_{\vec{u},\vec{w}}$.

3.4 Improved Sign Determination Algorithm

We can now run a variant of the basic algorithm using the compact sets. In the initial phase of the improved algorithm, the leaf nodes are assigned $C_0[i] = \{2\lambda^\dagger(i,b)\}$ as $F_0[i] = \{\pm 2\lambda^\dagger(i,b)\}$, $0 \le i < 2^n$. In the next layer, $C_1[i] = \{(\lambda^\dagger(i,b) + \lambda^\dagger(i+2^{n-1},b), \lambda^\dagger(i,b) - \lambda^\dagger(i+2^{n-1},b))\}$, $0 \le i < 2^n$. In the $(\ell+1)$-th layer, $C_{\ell+1}[i]$ is constructed from $C_\ell[i]$ and $C_\ell[i+2^{n-\ell-1}]$, $1 \le \ell < n$ and $0 \le i < 2^{n-\ell-1}$. After n iterations, the solutions to

[5]We only analyze the positive case. The arguments for the negative case are similar.

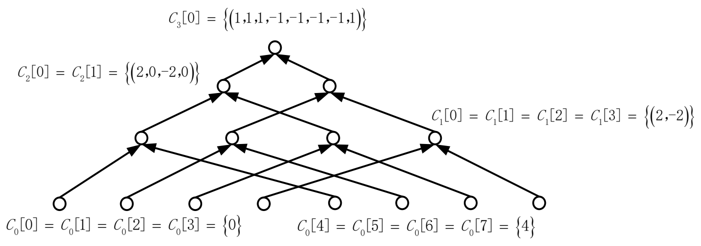
$C_3[0] = \{(1,1,1,-1,-1,-1,-1,1)\}$
$C_2[0] = C_2[1] = \{(2,0,-2,0)\}$
$C_1[0] = C_1[1] = C_1[2] = C_1[3] = \{(2,-2)\}$
$C_0[0] = C_0[1] = C_0[2] = C_0[3] = \{0\}$
$C_0[4] = C_0[5] = C_0[6] = C_0[7] = \{4\}$

Figure 2: The Tree Structure for a Sign Determination Problem

the sign determination problem are the vectors in the full set $F_n[0]$, which can be easily reconstructed from the compact set $C_n[0]$ by Equation 9. The search process of the sign determination problem using compact sets is stated in Algorithm 3.

We show an example with $\vec{\lambda}_b^\dagger = (0,0,0,0,2,2,2,2)^T$ to contrast the basic algorithm with the improved strategy. We apply Algorithm 3 to solve the sign determination problems and show the tree structure involved in solving the sign determination problem in Figure 2. Note that the compact sets in each layer are stored in the corresponding leaf nodes. The compact set technique is shown by constructing $C_3[0]$ from $C_2[0] = C_2[1] = \{(2,0,-2,0)\}$. We denote $\vec{v} = (2,0,-2,0)$. It can be seen that $C_2[0]$ is 1-symmetric, i.e., $J = \{1\}$. Algorithm 2 is applied to construct $M_{\vec{v},\vec{v}}$, where $M_{\vec{v},\vec{v}} = \{(2,0,-2,0),(0,2,0,-2)\}$. To build $C_3[0]$, we compute $E_2((2,0,-2,0),(2,0,-2,0)) = (2,0,0,0,-2,0,0,0)$ and $E_2((2,0,-2,0),(0,2,0,-2)) = (1,1,1,-1,-1,-1,-1,1)$. It is obvious that $C_3[0] = \{(1,1,1,-1,-1,-1,-1,1)\}$.

Note that when the basic algorithm is applied, the full set $F_3[0]$ is constructed from the full sets $F_2[0]$ and $F_2[1]$, where $F_2[0] = F_2[1] = \{(2,0,-2,0),(-2,0,2,0),(0,2,0,-2),(0,-2,0,-2)\}$. For each $\vec{u} \in F_2[0]$ and $\vec{v} \in F_2[1]$, we compute $E_2(\vec{u},\vec{v})$ and obtain 16 elements in the full set $F_3[0]$, whereas the compact set $C_3[0]$ contains only one element. To obtain the full set $F_3[0]$, we only apply simple permutations on the elements of $C_3[0]$, which avoids repeated computations. Thus, it can be concluded that applying the compact sets in the reconstruction procedure can save both time and memory complexity compared with the basic algorithm. We note that the advantage of applying the compact sets is more significant as the size of the full set is larger.

The number of the solutions of its sign determination problem is equal to the size of the Boolean functions which are DDT-equivalent to $b \cdot S(x)$, $1 \le b < 2^m$. The Boolean functions corresponding to the solutions of its sign determination problem share the same squared LAT with $b \cdot S(x)$, i.e., $(\vec{\lambda}_0^\dagger, \vec{\lambda}_b^\dagger)$. These Boolean functions are DDT-equivalent with $b \cdot S(x)$. When $b \cdot S(x)$ has nontrivial DDT-equivalence classes, $T_n[0]$ contains multiple vectors.

Given enough memory, Algorithm 3 can solve all the sign determination problems. However, for some instances, the amount of vectors in the internal layer grows sharply, which demands too much memory. In this situation, a threshold H on the number of internal vectors can be preset heuristically with respect to the accessible memory of the attacker. In the ℓ-th layer, if the size of $C_\ell[i]$ rises above the threshold H, the search process is interrupted, where $0 \le \ell < n$ and $0 \le i < 2^{n-\ell}$.

We call a column in the absolute LAT *good* if it can be recovered under the threshold H applying Algorithm 3; otherwise *bad*. In some cases, there exist both good columns and bad columns in the absolute LAT. For example, the S-boxes of CAST-256 [Ada99], like $S0$, are 8×32 S-boxes, which are constructed by choosing 32 distinct bent functions as the components (see [Ada97] for details). It indicates that each entry of the columns $\{\vec{\lambda}_{2^i}^\dagger | 0 \le i < 32\}$ is $2^{8/2-1} = 8$, i.e., all the entries of the LAT columns are ± 8. It has

Algorithm 3 Improved Algorithm for Solving the Sign Determination Problem

```
1: **Input:** $\vec{\lambda}_b^{\dagger}$;
2: **Output:** $F = \{\vec{u} | H_n \vec{u} = 2\vec{\lambda}_b, \vec{u}[0] = 1\}$
3: **for** each integer $i \in [0, 2^n - 1]$ **do**
4:     $C_0[i] = \{2\lambda^{\dagger}(i, b)\}$                                    ▷ Initialization
5: **end for**
6: $C_n[0] = \text{LAYER}(C_0, 0)$
7: Construct the full set $F_n[0]$ from $C_n[0]$.
8: **return** $F = \{\vec{u} | \vec{u} \in F_n[0], \vec{u}[0] = 1\}$.
9:
10: **procedure** LAYER($C_\ell$, $\ell$);
11:     **for** each integer $i \in [0, 2^{n-\ell-1} - 1]$ **do**
12:         **if** there are no vectors in $C_\ell[i]$ or $C_\ell[i + 2^{n-\ell-1}]$ **then**
13:             **return There exist no S-boxes corresponding to the given DDT!**
14:         **end if**
15:         $C_{\ell+1}[i] = \emptyset$
16:         Randomly pick a vector from $C_\ell[i]$ and compute $J = \{j \big| C_\ell[i]$ is $j$-symmetric,
    $0 \leq j < \ell\}$                                                      ▷ Theorem 3
17:         **for** each $\vec{w}$ in $C_\ell[i + 2^{n-\ell-1}]$ **do**
18:             **for** each $\vec{u}$ in $C_\ell[i]$ **do**
19:                 $M = \text{CONSTRUCTSET}(\vec{u}, [\vec{w}]^+, J)$
20:                 **for** each $\vec{v}$ in $M$ **do**
21:                     $\vec{r} = E_\ell(\vec{u}, \vec{v})$
22:                     **if** $\ell < n$ **then**
23:                         **if** every entry in $\vec{r}$ is even and $[-2^{n-\ell-1}, 2^{n-\ell-1}]$ **then**
24:                             $C_{\ell+1}[i] = C_{\ell+1}[i] \cup \{\vec{r}\}$
25:                         **else**
26:                             Discard $\vec{r}$
27:                         **end if**
28:                     **else**
29:                         **if** every entry in $\vec{r}$ is 1 or $-1$ **then**        ▷ when $\ell = n$
30:                             $C_n[i] = C_n[i] \cup \{\vec{r}\}$
31:                         **else**
32:                             Discard $\vec{r}$
33:                         **end if**
34:                     **end if**
35:                 **end for**
36:             **end for**
37:         **end for**
38:     **end for**
39:     **if** $\ell < n$ **then**
40:         LAYER($C_{\ell+1}$,$\ell + 1$)
41:     **else**
42:         **return** $C_n[0]$
43:     **end if**
44: **end procedure**
```

been proven by Langevin and Leander in [LL11] that the number of bent functions in dimension eight is approximately 2^{106}. Thus, the sign determination problem for the columns $\{\vec{\lambda}^{\dagger}_{2^i} | 0 \leq i < 32\}$ is too computationally expensive to be solved, i.e., these columns are bad columns. However, there are still some good columns in the absolute LAT of CAST-256's $S0$ if the attacker sets the threshold to 2000. For example, $\vec{\lambda}^{\dagger}_6$ and $\vec{\lambda}^{\dagger}_7$ corresponding to the 6th and 7th columns of its LAT.

According to our experiments with input size n between 8 and 14, the number of solutions for the good columns is no more than 2^{n+2}, i.e., $T_n[0]$ contains at most two vectors. We note that determining the size of the DDT-equivalence classes of a Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ is still an open problem and determining a suitable H, or even telling in advance whether a column is good, is also an open problem.

3.5 Heuristic Analysis of Time and Memory Complexities

We now analyze the memory complexity of Algorithm 3. In the ℓ-th layer, there are $2^{n-\ell}$ nodes in the tree structure, $0 \leq \ell \leq n$. Each node contains at most H vectors of length 2^ℓ and the entry of the vector ranges from $2^{n-\ell}$ to $-2^{n-\ell}$. The memory complexity of storing the nodes in the ℓ-th layer is $O(H \cdot (n - \ell + 1) \cdot 2^n)$ bits. For each $0 \leq i < 2^{n-\ell-1}$, when constructing $C_{\ell+1}[i]$, $M_{\vec{u},\vec{w}}$ contains at most 2^ℓ vectors with length of 2^ℓ, where $\vec{u} \in C_\ell[i]$ and $\vec{w} \in C_\ell[i + 2^{n-\ell-1}]$. Thus, $O((n - \ell + 1)2^{2\ell})$ bits of memory are needed to store $M_{\vec{u},\vec{w}}$, To conclude, the memory complexity of Algorithm 3 is $O(H \cdot n^2 2^n + n 2^{2n})$ bits.

Similarly, we analyze the time complexity of Algorithm 3. To construct $C_{\ell+1}[i]$, the attacker needs to apply Algorithm 2 for constructing $M_{\vec{u},\vec{w}}$ first for each $\vec{u} \in C_\ell[i]$ and $\vec{w} \in C_\ell[i + 2^{n-\ell-1}]$. Note that the complexity of this step is negligible as the attacker only applies permutations on vectors. Then, the attacker computes $E_\ell(\vec{u}, \vec{v})$ for each vector $\vec{u} \in C_\ell[i]$ and $\vec{v}$ in $M_{\vec{u},\vec{w}}$. The complexity of constructing $C_{\ell+1}[i]$ is thus at most $O(H \cdot H 2^{n-\ell} \cdot 2^{n-\ell}) = O(H^2 2^{2(n-\ell)})$. The time complexity of constructing $C_{\ell+1}$ is no more than $O(H^2 2^{3(n-\ell)})$. Thus, the upper bound of the time complexity is $O(H^2 2^{3n})$.

4 Applying Algorithm 3 for Reconstructing the S-box

The procedure of reconstructing an $n \times m$ S-box is related to the number of good columns defined in Section 3.4. We suppose that the attacker has solved the sign determination problem for k independent good columns, $1 \leq k \leq m$. In the sign determination problem for the c_i-th column, the possible candidates for the Boolean function $c_i S(x)$ are recovered by Algorithm 3. We call it the matching phase for the k good columns when the combination of these candidates is searched with respect to the squared LAT, $1 < k \leq m$.

After the matching phase for the k good columns, the Boolean functions $c_0 S(x)$, $\cdots$, $c_{k-1} S(x)$ are obtained. As mentioned before, when $k = m$, the attacker can reconstruct the S-box using linear algebra. When $k < m$, applying the knowledge of $c_0 S(x)$, $\cdots$, $c_{k-1} S(x)$, we propose a new technique that improves the guess-and-determine algorithm of [BCJS19].

4.1 The Matching Phase for the k Good Columns

Let V_i be the set which contains the output vectors from Algorithm 3 with respect to the c_i-th squared LAT column, where $0 \leq i < k$. In the matching phase, the Boolean functions $c_0 S(x)$, $\cdots$, $c_{k-1} S(x)$ are obtained by searching the vectors in V_i to match the squared LAT applying a basic property of the Hadamard product.

Property 3. For any $0 \leq b, c < 2^n$,

1. $\vec{s}_{b \oplus c} = \vec{s}_b \odot \vec{s}_c$.

2. $\pi_j^n(\vec{s}_{b\oplus c}) = \pi_j^n(\vec{s}_b) \odot \pi_j^n(\vec{s}_c)$, $0 \leq j < n$.

Property 3 is obvious from the definition of $\vec{s}_b$ and the Hadamard product. Combining the first formula in Property 3 with Property 2, we obtain that the $(b \oplus c)$-th column $\vec{\lambda}_{b\oplus c}$ in the LAT can be deduced by $1/2H_n \cdot \vec{s}_{b\oplus c} = 1/2H_n \cdot (\vec{s}_b \odot \vec{s}_c)$. For each two vectors $\vec{u} \in V_i$ and $\vec{v} \in V_j$, the attacker computes a new vector $\vec{w} = 1/2H_n \cdot (\vec{u} \odot \vec{v})$. Then, the attacker can easily detect whether $\vec{u}$ and $\vec{v}$ are consistent with the squared LAT by verifying whether $\vec{w}^\dagger = \vec{\lambda}^\dagger_{b\oplus c}$. We call $\vec{u}$ and $\vec{v}$ a matching vector pair if they are consistent with the absolute LAT column $\vec{\lambda}^\dagger_{b\oplus c}$.

Now we discuss the matching phase of the c_i-th column and the c_j-th column, $0 \leq i < j < k$. It should be noted that it is not necessary to verify the match for every pair of vectors from V_i and V_j. In the reconstruction problem, our purpose is to find a representative $S(x)$ in the equivalence class $\{S(x \oplus c) \oplus d | c \in \mathbb{F}_2^n, d \in \mathbb{F}_2^m\}$. For example, when the matching phase begins with the c_0-th and c_1-th columns, let us assume that there are q distinct symmetric-equivalence classes in the solution of the sign determination problem of the c_0-th column, i.e., $V_0 = \{\vec{v} | \vec{v} \in [\vec{u}_p], 0 \leq p < q\}$. The set of vector pairs which needs to be tested is $\{(\vec{u}_p, \vec{w}) | 0 \leq p < q, \vec{w} \in V_1\}$. When the attacker finds the representatives of matching vector pairs in $V_0 \times V_1$ that are consistent with the squared LAT, i.e., $\{(\vec{u}_{p_0}, \vec{w}) | 0 \leq p_0 < q, \vec{w} \in V_1\}$, the other matching vector pairs in $V_0 \times V_1$ can be constructed by the second formula in Property 3. Similarly, once the attacker obtains $c_0 S(x)$ and $c_1 S(x)$ corresponding to the matching vector pairs, all other Boolean functions can be recovered by the translation $c_0 S(x \oplus c) \oplus d$ and $c_1 S(x \oplus c) \oplus d$ following Property 1.

The number of the matching vector pairs between V_i and V_j is related to the number of the Boolean functions which are DDT-equivalent to $(c_i S(x), c_j S(x))$. More precisely, the matching phase over V_i and V_j finds the vectorial Boolean function $G(x)$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^2$, whose absolute LAT is $(\vec{\lambda}^\dagger_0, \vec{\lambda}^\dagger_{c_i}, \vec{\lambda}^\dagger_{c_j}, \vec{\lambda}^\dagger_{c_i \oplus c_j})$. Thus, $G(x)$ shares the same DDT with $(c_i S(x), c_j S(x))$. Note that the problem of determining the size of DDT-equivalence class of a Boolean function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^2$ is also an open issue.

As the size of DDT-equivalence class is unknown, we restrict the prescribed DDT to be a family of S-boxes for which the DDT-equivalence class is trivial according to the following conjecture proposed in [BCJS19].

Conjecture 1. *Suppose that S is a permutation over $\mathbb{F}_2^n$ and the rows of the DDT of S are pairwise distinct. Then, the DDT-equivalence class of S is trivial, i.e., only contains the permutations of the form $S(x \oplus c) \oplus d$, where $c, d \in \mathbb{F}_2^n$.*

The matching phase for k good columns is shown in Algorithm 4 repeating the matching phase of the i-th good column and the $(i+1)$-th good column, $0 \leq i \leq k-2$. For the S-boxes with trivial DDT-equivalence class, one combination is expected to be returned from Algorithm 4. If Conjecture 1 does not hold when the DDT-equivalence class of S is nontrivial, lines 9 and 17 in Algorithm 4 should be removed and the search continues with a set of the match vector pairs.

In our case, the number of solutions for good columns is $O(2^n)$. The time complexity of Algorithm 4 is $O((|V_1| + \cdots + |V_{k-1}|)2^{2n}) = O(k2^{3n})$. The memory complexity is negligible.

4.2 The Improved Guess-and-Determine Algorithm

Now we suppose that the attacker has obtained k $(1 \leq k < m)$ Boolean functions, i.e., $c_0 S(x)$, ..., $c_{k-1} S(x)$, using Algorithm 4. We present an improved GD algorithm that takes the DDT table and the k Boolean functions as its inputs and returns a representative of the DDT-equivalence class.

The improved GD algorithm implements the tree-traversal structure of [BCJS19]. The improved GD algorithm begins by fixing $S(0)$ to be zero in the initial layer. In the i-th layer,

Algorithm 4 The Matching Phase Given k Good Columns

```
1: Input: the index set of the good columns $C = \{c_0, \ldots, c_{k-1}\}$, the corresponding
   solution sets $V_0, \ldots, V_{k-1}$ and the squared LAT;
2: Output: $c_0S(x), \ldots, c_{k-1}S(x)$;
3: for each $i \in [0, k-2]$ do
4:     if $i = 0$ then
5:         for each $\vec{u} \in \{\vec{u}_0, \ldots, \vec{u}_p\}$ and $\vec{v} \in V_1$ do
6:             $\vec{w} = 1/2H_n \cdot (\vec{u} \odot \vec{v})$
7:             if $\vec{w}^\dagger = \vec{\lambda}^\dagger_{c_i \oplus c_{i+1}}$ then
8:                 $\vec{p}_0 = \vec{u}, \vec{p}_1 = \vec{v}$
9:                 break        ▷ this line is to be removed if the DDT-equivalence class is
   nontrivial.
10:            end if
11:        end for
12:    else
13:        for each $\vec{v} \in V_{i+1}$ do
14:            $\vec{w} = 1/2H_n \cdot (\vec{p}_i \odot \vec{v})$
15:            if $\vec{w}^\dagger = \vec{\lambda}^\dagger_{c_i \oplus c_{i+1}}$ then
16:                $\vec{p}_{i+1} = \vec{v}$
17:                break        ▷ this line is to be removed if the DDT-equivalence class is
   nontrivial.
18:            end if
19:        end for
20:    end if
21: end for
22: Deduce $c_0S(x), \ldots, c_{k-1}S(x)$ from $\vec{p}_0, \ldots, \vec{p}_{k-1}$
23: return $c_0S(x), \ldots, c_{k-1}S(x)$.
```

the algorithm determines the possible assignments for $S(i)$, $i = 1, \ldots, 2^n - 1$, by checking the constraints imposed by the DDT. We follow the notations from [BCJS19] by denoting the set of possible values for $S(i)$ by $\mathcal{R}_i = \{y \big| \delta(i, y) \neq 0\}$ imposed by the given DDT. It implies that $S(i)$ is in the set $\mathcal{L} = \{x \oplus S(0) | x \in \mathcal{R}_i\} \cap \cdots \cap \{x \oplus S(i-1) | x \in \mathcal{R}_{i \oplus (i-1)}\}$.

In our approach, the knowledge of $c_0S(x)$, …, and $c_{k-1}S(x)$ reduces the size of the set $\mathcal{L}$. For every element $x \in \mathcal{L}$, if any of the equalities $c_0x = c_0S(i)$, $\cdots$, $c_{k-1}x = c_{k-1}S(i)$ does not hold, x is removed from $\mathcal{L}$. Then the guess and determine of [BCJS19] is applied with the reduced lists. The reconstruction process is illustrated in a recursive way in Algorithm 5.

Next, we analyze the time complexity of Algorithm 5 on a random S-box for $1 \leq k < m$. The analysis of the original GD algorithm when $k = 0$ is presented in Appendix A. In the first layer, after discarding the non-consistent values of $S(1)$ based on the DDT, there are $2^m P^{DDT}_{n,m}$ possible values on average. Similarly, after checking the constraints imposed by $c_0S(x)$, $\cdots$, and $c_{k-1}S(x)$, there are $2^{m-k} P^{DDT}_{n,m}$ possible values left for $S(1)$. By the i-th layer, the above process is repeated and the number of the possible assignments $S(1), \cdots, S(i)$ on average is

$$W_i = \begin{cases} 2^{(m-k)i} (P^{DDT}_{n,m})^{\frac{i^2+i}{2}} & ,0 \leq i \leq K, \\ 1 & ,K < i < 2^n, \end{cases}$$

where K is the smallest positive integer such that $2^{(m-k)i} (P^{DDT}_{n,m})^{\frac{i^2+i}{2}} < 1$.

In the $(i+1)$-th layer, there are $2^m P^{DDT}_{n,m}$ possible assignments for $S(i+1)$, where $i < K$. For each possible assignment, the attacker checks whether $S(i+1) \oplus S(1), \ldots, S(i+1) \oplus S(i)$

Algorithm 5 The Improved Guess-and-Determine Algorithm

```
1: **Input:** the indices of good columns $c_0, \ldots, c_{k-1}$, the Boolean functions $c_0S(x), \cdots, c_{k-1}S(x)$ and the given DDT
2: **Output:** one representative in the DDT-equivalence class
3: $\vec{s}$ is initialized as a vector of $2^m$ zeros.
4: ImprovedGD($\vec{s}$, 1)
5: **return** $\vec{s}$
6:
7: **procedure** ImprovedGD($\vec{s}$,$i$)
8:     **if** $i < 2^m$ **then**
9:         $\mathcal{L} = \bigcap_{0 \le j < i} \{x \oplus \vec{s}\,[j] | x \in \mathcal{R}_{i \oplus j}, c_0S(i) = c_0 \cdot x, \cdots, c_{k-1}S(i) = c_{k-1} \cdot x\}$
10:    **else**
11:        **if** the DDT of $\vec{s}$ matches the given DDT **then**
12:            **return** $\vec{s}$
13:        **end if**
14:    **end if**
15:    **if** $L \neq \emptyset$ **then**
16:        **for** each $x \in \mathcal{L}$ **do**
17:            $\vec{s}\,[i] = x$
18:            ImprovedGD($\vec{s}$,$i + 1$)
19:        **end for**
20:    **else**
21:        **return There exist no S-boxes corresponding to the given DDT!**
22:    **end if**
23: **end procedure**
```

are consistent with the DDT. The expected complexity of this process is $1 + P^{DDT}_{n,m} + \cdots + (P^{DDT}_{n,m})^i < 2$ tests. There are $(P^{DDT}_{n,m})^{i+1} \cdot 2^m W_i(k)$ possible assignments for $S(1), \cdots, S(i+1)$ at this stage. Each assignment should be tested with respect to the constraints $c_0S(i+1), \cdots$, and $c_{k-1}S(i+1)$. The number of checks on each assignment is also no more than 2. Thus, the time complexity of this layer is $2^{m+1}P^{DDT}_{n,m} \cdot W_i(k) + 2^{m+1}(P^{DDT}_{n,m})^{i+1}W_i(k) \approx 2^{m+1} \cdot P^{DDT}_{n,m} \cdot W_i(k)$.

From the K-th layer, $W_i(k) = 1$ and the time complexity of each layer is no more than $2^{m+1}P^{DDT}_{n,m}$. Thus, the expected time complexity of Algorithm 5 is

$$T_{n,m}(k) = 2^{m+1}P^{DDT}_{n,m} \sum_{i=0}^{2^n-2} W_i(k).$$

We evaluate the time complexity for the original guess and determine algorithm for $n = 8$ with different values of m, which is shown in Table 1. It should be noted that increasing the size of the output of the S-box (i.e., n) makes the reconstruction process easier. Thus, an $n \times m$ S-box with $m \gg n$ is not a significantly secure option when designing a secret non-linear layer for a cryptographic primitive.

Recall that from Equation 3, the time complexity of deducing a column of the absolute LAT from the DDT of an 8-bit S-box is about $2^{24}P^{DDT}_{n,m} \approx 2^{23.28}$, which is greater than the complexity of the original GD algorithm of $2^{22.34}$. Thus, for a random 8-bit S-box, it is better to apply the original GD algorithm when reconstructing the S-box from its DDT.

Table 1: $\log_2 T_{n,m}(0)$ for random S-box $n = 8$ with different m

m	8	9	10	11	12
$\log_2 T_{n,m}(0)$	22.34	17.12	16.11	15.99	15.98

We also evaluate the time complexity for the GD phase for n-bit S-box with different k, where $9 \leq n \leq 14$. The results are shown in Table 2. It is obvious that to optimize the original GD algorithm, the attacker should find at least two independent good columns. It should be noted that from Table 2, the original GD algorithm ($k = 0$) quickly becomes impractical with the size of S-box growing. For example, reconstructing a 14-bit S-box with the original GD algorithm is infeasible with the expected time complexity of about $2^{68.37}$; whereas for $k \geq 9$ it is no more than $2^{27.68}$. Hence, given enough good columns, our technique improves the original GD algorithm and makes the reconstruct procedure practical to be implemented.

Table 2: $\log_2 T_{n,n}(k)$ for random S-box $9 \leq n \leq 13$ with Different k

n \ k	0	1	2	3	4	5	6
9	28.14	29.98	24.91	20.75	18.34	17.75	17.67
10	34.7	36.79	30.98	25.92	21.91	20.04	19.71
11	42	44.35	37.79	31.97	26.94	23.18	21.86
12	50.05	52.65	45.35	38.8	32.98	27.98	24.61
13	58.84	61.7	53.66	46.35	39.8	33.98	29.05
14	68.37	71.49	62.7	54.66	47.35	40.79	34.99

n \ k	7	8	9	10	11	12	13
9	17.65	17.65	-	-	-	-	-
10	19.66	19.66	19.65	-	-	-	-
11	21.68	21.66	21.65	21.65	-	-	-
12	23.76	23.67	23.66	23.65	23.65	-	-
13	26.21	25.71	25.66	25.66	25.65	25.65	-
14	30.18	27.96	27.68	27.66	27.65	27.65	27.65

5 Experiments

We verify our results by implementing our reconstruction technique on random S-boxes, the S-boxes of some existing block ciphers, 4-differential uniform permutations, and APN functions. Our experiments are implemented in C++ using a g++ compiler with -O2 optimization with a single core of an Intel(R) Xeon(R) E5-2620 v3 CPU @ 2.40GHz of 64GB memory. The related codes are available at `https://github.com/xiaohuangthu/sbox`.

5.1 Random S-boxes

For each $8 \leq n \leq 14$, we compare the performance of the GD algorithm and our approach by implementing the two methods on 100 random n-bit S-boxes. For $8 \leq n \leq 12$, we set the threshold H to be 2000. The necessary memory for $8 \leq n \leq 12$ is 4.2MB, 10.4MB, 26.9MB, 67.7MB, and 172.6MB, respectively according to our analysis in Section 3.5. For the 13-bit instances, when $H = 2000$, there are not enough good columns found. Thus, the threshold is increased to 6000 for the random 13-bit S-boxes, which costs 1.2GB memory at most. We set $H = 12000$ for the same reason when reconstructing 14-bit cases, which needs 5.3GB memory at most. We also note that as shown in Table 2 when the number of good columns grows, the effect of reducing the search phase of the GD phase becomes less significant. In the experiments, we set the value of k to make the complexity of the GD phase practical, e.g., $k = 6$ for $n = 14$.

The running time of the GD algorithm and our approach is shown in Figure 3. We denote the running time as T. The time measurements of our approach include finding sufficient number of good columns and using them for recovering the S-box. The statistical data of the running time on the instances is presented in Table 3. The running time of the original GD algorithm for larger S-boxes (i.e., 12-, 13-, and 14-bit) is estimated based on the following approach:

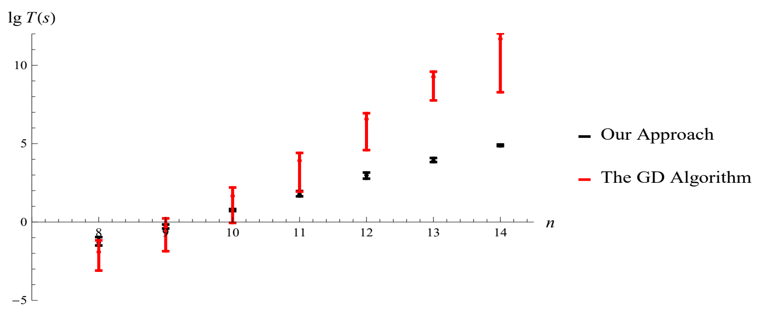

Figure 3: The Running Time on Random S-boxes

Table 3: The Statistical Data for The Instances

n	k	Min (s)	Max (s)	Average (s)	Median (s)	Standard Deviation	Method
8	0	8.01×10^{-4}	0.07	0.01	0.01	0.01	GD algorithm
8	2	0.03	0.11	0.05	0.05	0.01	Our Approach
9	0	0.01	1.70	0.49	0.05	0.42	GD algorithm
9	3	0.39	0.70	0.50	0.49	0.06	Our Approach
10	0	0.88	159.94	45.80	38.83	36.0	GD algorithm
10	3	4.98	6.74	5.48	5.45	0.32	Our Approach
11	0	86.97	2.56×10^4	8.20×10^3	7.00×10^3	6.26×10^3	GD algorithm
11	4	43.61	94.68	58.23	57.00	11.34	Our Approach
12	0	3.88×10^4	8.73×10^6	3.66×10^6	4.17×10^6	2.17×10^6	GD algorithm
12	4	584.22	1437.26	962.33	925.08	167.38	Our Approach
13	0	5.72×10^7	3.90×10^9	1.83×10^9	1.96×10^9	9.90×10^8	GD algorithm
13	6	6.68×10^3	1.22×10^4	8.07×10^3	8.04×10^3	878.56	Our Approach
14	0	1.90×10^8	1.09×10^{12}	4.79×10^{11}	4.78×10^{11}	2.88×10^{11}	GD algorithm
14	6	6.93×10^4	8.81×10^4	7.52×10^4	7.39×10^4	4.07×10^3	Our Approach

As mentioned, the time complexity of the GD algorithm on random 14-bit S-boxes is about $2^{68.27}$. Obviously, we have not run an experiment for that long. Indeed we estimate the running time of the GD algorithm using the following methodology: first, we fix $S(1)$, $\cdots$, $S(4)$, and $S(5)$ to the correct values and apply the GD algorithm on the remaining values, denoting the running time to be t_0. Then, we repeat the procedure with wrong assignments for $S(1)$, $\cdots$, $S(4)$, and $S(5)$ for 100 times. We denote the average running time for a wrong guess by t_1. Thus, if there are C assignments to check before the correct one for $S(1)$, $\cdots$, $S(4)$, and $S(5)$, then the estimated running time is $t_0 + C \cdot t_1$.

It can be seen from Figure 3 that the advantage of our approach over the GD algorithm sharply increases when the size of the S-box grows. Among 100 random 8-bit S-boxes, our approach is better than the GD algorithm in 2 cases. For the random 9-bit S-boxes, our approach is better in 44 cases. For the random 10-bit S-boxes, our approach is better in 87 instances. When the input size of S-boxes is larger than 11, our approach is better in all cases. For example, as shown in Table 3, the average running time of the GD algorithm on the random 14-bit S-boxes is approximately 15178.9 years. The average running time of our approach is 7.52×10^4s, which is less than one day. It can be seen from the standard deviation in Table 3 that the running time of our approach is more stable than the GD algorithm.

5.2 Specific S-boxes of Existing Ciphers

We also run experiments on the 8-bit S-boxes of several block ciphers, including AES [DR02], Camellia [AIK+01], SEED [IA], ARIA [KKP+04], SKIPJACK [Age], CLEFIA [SSA+07], and Streebog [oTRM]. In these experiments, Algorithm 3 is applied to solve the sign determination problems with the threshold preset to be 2000 and the number of good columns is set to 2. While for many of the tested S-boxes, we have found good columns; for the S-box $S0$ of CLEFIA, there exists no good column in its absolute LAT. The running

Table 4: The Running Time for Existing S-boxes

	AES	ARIA	Camellia	CLEFIA-$S0$	CLEFIA-$S1$
GD Algorithm	1.8s	1.88s	2.47s	0.004s	3.27s
Our Approach ($k = 2$)	0.07s	0.085s	0.188s	-	0.29s
	SEED-$S0$	SEED-$S1$	Skipjack	Streebog	
GD Algorithm	1.59s	9.19s	0.021s	0.049s	
Our Approach ($k = 2$)	0.15s	0.23s	0.063s	0.051s	

time of the experiments is shown in Table 4.

It can be seen from Table 4 that it is more effective to reconstruct the S-boxes of AES, ARIA, SEED, Camellia, and $S1$ of CLEFIA from their DDT by solving the sign determination problem of two independent columns and applying Algorithm 5 with the knowledge of two Boolean functions related to the S-box. For example, using the GD algorithm, the reconstruction procedure takes 9.19s to recover the $S1$ in SEED from its DDT. However, when the attacker solves the sign determination problem of two independent columns, the reconstruction costs only 0.23s. It should be noted that the S-boxes of AES, ARIA, SEED, Camellia and $S1$ of CLEFIA are of 4-differential uniformity.

For other 8-bit S-boxes in our experiments, i.e., the S-boxes of Streebog, Skipjack and $S0$ of CLEFIA, it is more effective to reconstruct the S-box from its DDT with the original GD algorithm of [BCJS19]. It should be noted that the S-boxes of Streebog, Skipjack and $S0$ in CLEFIA are 8-, 12-, and 10-differential uniformity, respectively.

5.3 4-differential uniformity S-boxes and APN functions

We applied our algorithms on some 4-differential uniformity permutations in Table 1 of [BCC10] for the input size between 9 and 14. The threshold is set to 12000. Although it is difficult to reconstruct the S-boxes with low differential uniformity according to our experiments, we can still find good columns in the absolute LAT of the 10-bit and 14-bit inverse functions, respectively. For example, there are 3 good columns found in the absolute LAT of the 14-bit inverse function, which reduce the searching space of guess-and-determine algorithm sharply.

It is hard to find good columns in the absolute LAT of APN functions. We applied our technique to the 7-bit S-box $S7$ and 9-bit S-box $S9$ in the block ciphers KASUMI [KAS], MISTY1 [Mat97], which are designed to be the APN permutations. We found no good columns in the absolute LATs of KASUMI's $S7$ and $S9$ and MISTY1's $S7$ and $S9$ even when we set the threshold $H = 12000$. Then, with the same threshold, we applied our technique to the APN functions of the input size between 6 and 11 listed in Table 3 of [Sun17]. It is interesting to note that we find good columns only in the 6-bit APN functions, the 8-bit Kasami function and the inverse functions with 7-bit, 9-bit and 11-bit input, respectively. Hence for such functions it seems that the standard GD algorithm in [BCJS19] is better than ours.

6 Conclusions

In this paper we presented a new algorithm for reconstructing an S-box from its DDT. The new algorithm is more efficient than the guess-and-determine algorithm proposed by Boura et al. in [BCJS19], for random S-boxes starting at the size of 10 bits, it outperforms the previous GD algorithm by several orders of magnitude.

Most notably, the new algorithm can be useful to explore problems related to DDTs. This includes theoretical explorations (e.g., whether there are two DDT-equivalent bijective S-boxes which are not linearly equivalent) or even the ability to construct an S-box from a "made up" DDT (i.e., picking the DDT and then constructing an S-box out of it), thus extending the analysis of Biryukov and Perrin in [BP15] for partial DDT constraints. We note that this capability may be used for designing stronger S-boxes with a 0 in selected places of the DDT. This would allow, for example, to make sure that differences which are optimal with respect to the linear layers (e.g., activate less S-boxes in non-full diffusion layers) cannot "co-exist" through the S-box itself.

Another related open problems are the problems of reconstructing an S-box from its *Boomerang Connectivity Table*, introduced in [CHP+18] and its *Differential-Linear Connectivity Table*, introduced in [BODKW19], respectively. These two tables are useful for evaluating the boomerang attack [Wag99] and the differential-linear attack [LH94], respectively. Both tables are related to the DDT. Hence, while at the moment there are no attacks that recover the BCT and DLCT of an unknown S-box, requiring the ability of reconstructing an S-box from them, this ability to reconstruct may help in exploring the properties of BCTs and DLCTs.

Acknowledgments

The authors thank Itai Dinur for his thoughtful insights. And the authors also thank Danny Keller for his implementation skills and efforts.

The research in this paper supported in part by the Israel Ministry of Science and Technology, the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by the Israeli Science Foundation through grant No. 880/18. Senyang Huang was supported in part by the Israeli Science Foundation through grant No. 573/16.

References

[Ada97] Carlisle M. Adams. Constructing Symmetric Ciphers Using the CAST Design Procedure. *Designs, Codes and Cryptography*, 12(3):283–316, Nov 1997.

[Ada99] Carlisle M. Adams. The CAST-256 Encryption Algorithm. 1999. https://tools.ietf.org/html/rfc2612, AES Candidate.

[Age] National Security Agency(NSA). SKIPJACK and KEA Algorithm Specifications.

[AIK+01] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis. In Douglas R. Stinson and Stafford Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg, 2001.

[BCC10] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential Properties of Power Functions. *IJICoT*, 1(2):149–170, 2010.

[BCJS19] Christina Boura, Anne Canteaut, Jérémy Jean, and Valentin Suder. Two Notions of Differential Equivalence on Sboxes. *Des. Codes Cryptography*, 87(2-3):185–202, 2019.

[BLN17] Céline Blondeau, Gregor Leander, and Kaisa Nyberg. Differential-Linear Cryptanalysis Revisited. *Journal of Cryptology*, 30(3):859–888, Jul 2017.

[BN13] Céline Blondeau and Kaisa Nyberg. New links between differential and linear cryptanalysis. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 388–404. Springer Berlin Heidelberg, 2013.

[BOBDK18] Achiya Bar-On, Eli Biham, Orr Dunkelman, and Nathan Keller. Efficient Slide Attacks. *Journal of Cryptology*, 31(3):641–670, Jul 2018.

[BODKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A New Tool for Differential-Linear Cryptanalysis. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 313–342, Cham, 2019. Springer Berlin Heidelberg.

[BP15] Alex Biryukov and Léo Perrin. On Reverse-Engineering S-Boxes with Hidden Design Criteria or Structure. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 116–140. Springer, 2015.

[BS91] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, Jan 1991.

[Car10] Claude Carlet. *Boolean Functions for Cryptography and Error-Correcting Codes*, page 257–397. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010.

[CHP$^+$18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang Connectivity Table: A New Cryptanalysis Tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714, Cham, 2018. Springer Berlin Heidelberg.

[CV95] Florent Chabaud and Serge Vaudenay. Links between Differential and Linear Cryptanalysis. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer Berlin Heidelberg, 1995.

[DGV95] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In Bart Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 275–285. Springer Berlin Heidelberg, 1995.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[DR07] Joan Daemen and Vincent Rijmen. Probability Distributions of Correlation and Differentials in Block Ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007.

[GOS98] GOST 28147-89. Cryptographic Protection for Data Processing Systems, Cryptographic Transformation Algorithm. *Government Standard of the U.S.S.R., Inv. No. 3583, UDC 681.325.6:006.354.*, 1998. (in Russian).

[IA] Korea Internet and Security Agency. SEED 128 Algorithm Specification. https://seed.kisa.or.kr/html/egovframework/iwt/ds/ko/ref/[2]_SEED+128_Specification_english_M.pdf.

[KAS] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, 3G Security, Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification, V3.1.1 (2001).

[KKP$^+$04] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. New Block Cipher: ARIA. In Jong-In Lim and Dong-Hoon Lee, editors, *Information Security and Cryptology - ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 432–445. Springer Berlin Heidelberg, 2004.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer Berlin Heidelberg, 1994.

[LL11] Philippe Langevin and Gregor Leander. Counting all bent functions in dimension eight 99270589265934370305785861242880. *Designs, Codes and Cryptography*, 59(1):193–205, Apr 2011.

[Mat94] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer Berlin Heidelberg, 1994.

[Mat97] Mitsuru Matsui. New Block Encryption Algorithm MISTY. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer Berlin Heidelberg, 1997.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-correcting Codes*, volume 16. Elsevier, 1977.

[Nyb94] Kaisa Nyberg. Differentially Uniform Mappings for Cryptography. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer Berlin Heidelberg, 1994.

[O'C94] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 360–370. Springer Berlin Heidelberg, 1994.

[O'C95] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. *Journal of Cryptology*, 8(2):67–86, Mar 1995.

[oTRM] Federal Agency on Technical Regulation and Metrology. GOST R 34.11-2012: Streebog hash function. https://www.streebog.net/.

[Sch94] Bruce Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In Ross Anderson, editor, *Fast Software Encryption*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer Berlin Heidelberg, 1994.

[SSA$^+$07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Alex Biryukov, editor, *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer Berlin Heidelberg, 2007.

[Sun17] Bo Sun. On Equivalence of Known Families of APN Functions in Small Dimensions. *CoRR*, abs/1709.07664, 2017.

[Wag99] David Wagner. The Boomerang Attack. In Lars Knudsen, editor, *Fast Software Encryption*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer Berlin Heidelberg, 1999.

A The Time Complexity of the Original Guess-and-Determine Algorithm

The original guess-and-determine algorithm in [BCJS19] also returns a representative S in the set $\{S(x \oplus c) \oplus d \big| c \in \mathbb{F}_2^n, d \in \mathbb{F}_2^m\}$. To achieve so, the attacker can fix $S(0)$ to be zero and fix $S(1)$ to be any value in $\mathcal{R}_i$. Thus, there is one possible case after the first layer. Similar to the analysis in Section 4.2, the number of the possible cases at the end of the i-th layer is

$$W_i = \begin{cases} 1 & ,i = 1, \\ 2^{m(i-1)}(P_{n,m}^{DDT})^{\frac{i^2+i-2}{2}} & ,2 \le i \le K, \\ 1 & ,K < i < 2^n, \end{cases}$$

where K is smallest positive integer such that $2^{m(i-1)}(P_{n,m}^{DDT})^{\frac{i^2+i-2}{2}} < 1$. In the $(i+1)$-th layer, the attacker need to check the consistency of $2^m P_{n,m}^{DDT} W_i$ cases with respect to the DDT. The complexity of the $(i+1)$-th layer is no more than $2^{m+1} P_{n,m}^{DDT} W_i$. As the algorithm starts from searching the assignment of $S(2)$, the time complexity of the original guess-and-determine algorithm is

$$T_{n,m}(0) = 2^{m+1} P_{n,m}^{DDT} \sum_{i=1}^{2^n-2} W_i.$$